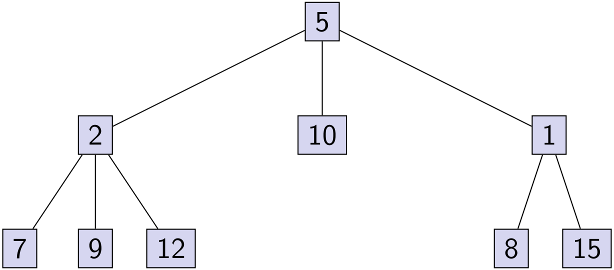
CMPT 280

Tutorial: m-ary Trees

Mark G. Eramian

University of Saskatchewan

General/m-ary Trees

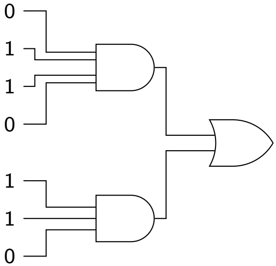


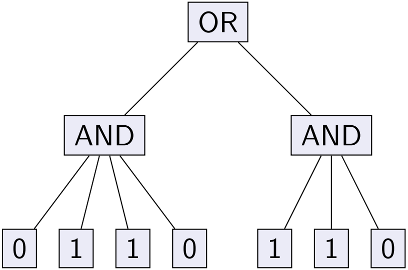
- A general tree is one in which each node may have any number of subtrees.
- An m-ary tree is one in which each node may have at most m subtrees.
- Each node must have a container to store subtrees (or at least their roots).

Example m-ary Trees

Logical Circuit

A logical circuit can be represented by an m-ary tree:

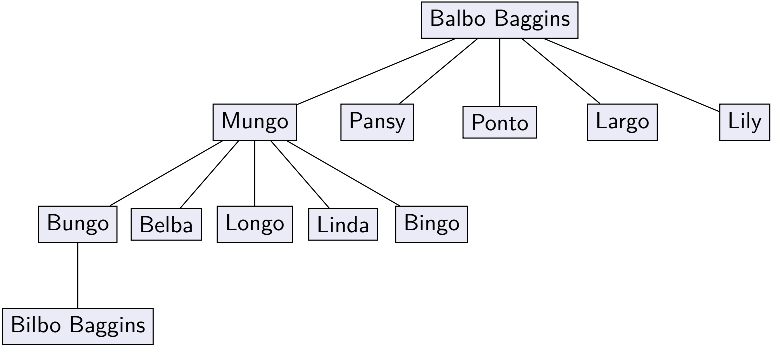




Example m-ary Trees

Genealogical Descendent Chart

This tree represents (some of) the descendants of Balbo Baggins, including his great-grandson, Bilbo Baggins, the well-known character from the works of J. R. R. Tolkein.



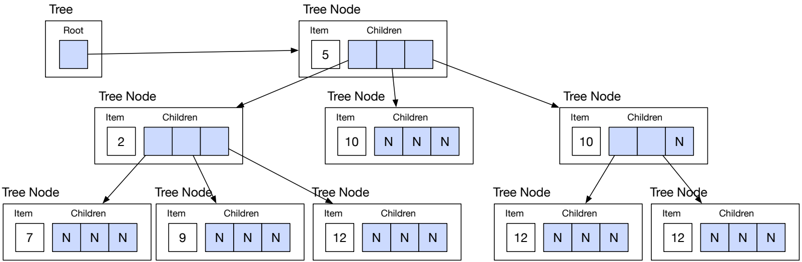
Implementing m-ary Trees

- In linked trees, each node must somehow store the references to its child nodes.
- We now examine two ways of implementing this storage.

General/m-ary Trees

What kind of container should be used to store the subtrees?

1. An array (N = null)



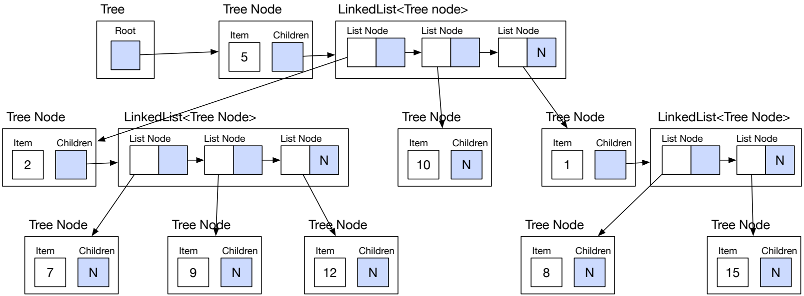
Pros: fast access to subtrees via indexing (time complexity?)

Cons: predetermined limit on number of children, potentially stores many null references (but not problematic for small m)

General/m-ary Trees

What kind of container should be used to store the subtrees?

2. A linked list (N = null)



Pros: no predetermined limit on the number of subtrees

Cons: slower access to specific children (time complexity?)

m-ary Trees in lib280

- The implementation of m-ary trees in `lib280` is the class `BasicMAryTree280<I>`. It can be found in the `lib280.tree` package.
- Nodes are of type `MAryNode280<I>`.
- Each node contains an **array** that stores its children.
- The capacity of these arrays are chosen when calling the constructor of the tree:

```java
// Create a tree containing with a single root node containing the string
// "Balbo Baggins" where each node can have up to six children.

BasicMAryTree<String> T = new BasicMAryTree<String>("Balbo Baggins", 6);
```

m-ary Trees in `lib280`

Important methods in `BasicMAryTree280<I>`:

rootItem(): get the item at the root of the tree

rootLastNonEmptyChild(): returns the integer index[1] of the last non-empty subtree of the root node.

rootSubtree(i): returns the `BasicMAryTree280` object that is the i-th subtree[1] of the root.

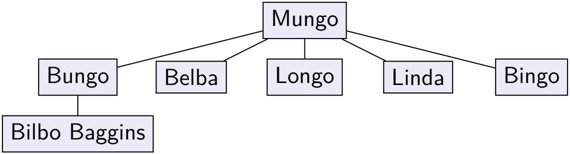
setRootSubtree(T, i): sets the i-th subtree[1] of the root node to be the `BasicMAryTree280` tree T.

And of course, there are many other methods that you can look at on your own.

[1] **Important note:** the index here starts at 1! If a tree's nodes can have six children than the first such child is the root of the subtree at index 1, and the last one is the subtree at index 6.

m-ary Trees in `lib280`

Here's how we could construct the genealogical descendent tree for the descendants of Mungo Baggins in the Baggins family tree:




```java
BasicMAryTree<String> T = new BasicMAryTree<String>("Mungo", 5);
T.setRootSubtree(new BasicMAryTree<String>("Bungo", 5), 1);
T.setRootSubtree(new BasicMAryTree<String>("Belba", 5), 2);
T.setRootSubtree(new BasicMAryTree<String>("Longo", 5), 3);
T.setRootSubtree(new BasicMAryTree<String>("Linda", 5), 4);
T.setRootSubtree(new BasicMAryTree<String>("Bingo", 5), 5);
BasicMAryTree<String> Bungo = T.rootSubtree(1);
Bungo.setRootSubtree(new BasicMAryTree<String>("Bilbo Baggins", 5), 1);
```

Recursive Traversal of m-ary Trees

Basic Idea:

```
Algorithm treeTraverse(curNode)
A recursive traversal of the tree that prints out the contents of each node.
Parameters:
curNode: reference to the current node in the traversal.

print curNode.item()

for each child i of curNode  //  i.e. for all the non-empty subtrees of curNode
    treeTraverse(curNode.getChildNode(i))
```

Recursive Traversal of m-ary Trees

Exercise

- Implement the algorithm on the previous slide for the `BasicMAryTree280<I>` class.
  - Public method to initiate traversal
  - Protected/private "helper" method for the actual recursion (rationale: algorithm requires node as parameter, but we do not want the nature of the nodes to be exposed in the public interface!)
- Trace through the implementation using the tree on slide 4 as input.